REMARKS

The Office Action of 06/04/2004 has been carefully considered. Reconsideration in view of the present Remarks is respectfully requested.

The present invention, in one aspect thereof, relates to a method of program execution in which, instead of repeatedly resolving symbolic code references, the results of such resolution are stored and used subsequently when the same symbolic code reference is later encountered. As set forth in claim 1, instructions are grouped into categories, and the results of symbol resolution are stored according to instruction category.

Claims 1-9 were rejected as being unpatentable over Darlet in view of Gee and further in view of Lai. Both Gee and Lai relate to run-time symbol resolution. Darlet, however, relates to a software loader and linker that operates prior to run-time but does not operate during run-time. The rejection states in part:

> [I]t would have been obvious...to modify the association reference/result structure as taught by Darlet...to implement...the grouping as taught by Gee.
>
> * * *
>
> [I]t would have been obvious...to implement the dynamic search, resolving and storing of the resulting operand numeric value by Lai [in combination with] Darlet/Gee's method.

Applicant respectfully disagrees. Despite superficial commonalities between the references, the references, the differences between the references far outweigh the similarities so as to negate any inference of obviousness.

Darlet teaches a process of "out-of-order" linking. A software module may be loaded that contains a symbolic reference that has not been resolved. In this instance, a default address value is substituted, the address value being such as to generate an exception upon execution. The symbolic reference is designated as being "pending."

Assuming a later-loaded software module causes the unresolved symbol reference to now be resolved, the newly-resolved address-value is substituted for the default address value.

The Gee reference describes something quite different. In Gee, when a software module (software object) is first invoked, symbolic references within the object are resolved and resulting values are stored within the software object itself (col. 22-23).

Regarding the Indexed CSA Area (Class Static Area) of Figure 3 of Gee, there is no teaching or suggestion in Gee of using such area to aid in or accelerate symbol resolution. The proposed combination of Gee with the primary reference for this purpose is therefore the product of impermissible hindsight and does not find any basis in the references themselves.

The further combination of Lai with Darlet/Gee is similarly difficult to justify. Whereas Darlet relates to pre-run-time linking, Lai relates to run-time symbol resolution. Motivation is lacking from the references themselves for one of ordinary skill in the art to combine the teachings of the references. Gee and Lai address the same problem, but address the problem in distinctly different ways. Whereas Gee stores resolved symbol references within the same software module (object) within which the reference occurs, Lai stores such references in a dynamic numeric reference buffer. Because of these distinct differences, a combination based on both Gee and Lee would not have been obvious to one of ordinary skill in the art.

Accordingly, the claims are believed to patentably define over the cited references. Notice of the same is respectfully requested.

Respectfully submitted,

Michael J. Ure, Reg. 33,089

Dated: September 22, 2004